ABSTRACT

The impact of various cache replacement policies act as the main deciding factor of system performance and efficiency in Chip Multi-core Processors. It is imperative for any level of cache memory in a multi-core architecture to have a well-defined, dynamic replacement algorithm in place to ensure consistent superlative performance. Many existing cache replacement policies such as Least Recently Used (LRU) policy etc have proved to work well in the shared Level 2 (L2) cache for most of the data set patterns generated by single threaded applications. But when it comes to parallel multi-threaded applications that generate differing patterns of workload at different intervals, the conventional replacement policies can prove sub-optimal as they generally do not abide by the spatial and temporal locality theories and do not acquaint dynamically to the changes in the workload.

This dissertation proposes three novel cache replacement policies for L2 cache that are targeted towards such applications. Context Based Data Pattern Exploitation Technique assigns a counter for every block of the L2 cache for monitoring the data access patterns of various threads and modifies the counter values appropriately to maximize the performance. Logical Cache Partitioning technique logically partitions the cache elements into four zones based on their 'likeliness' to be referenced by the processor in the near future. Replacement candidates are chosen from the zones in the ascending order of their 'likeliness factor' to minimize the number of cache misses. Sharing and Hit based Prioritizing replacement technique takes the sharing status of the data elements into consideration while making replacement decisions. This information is combined along with the number of hits received by the element to embark upon a priority based on which the replacement decisions are made.

The proposed techniques are effective at improving cache performance and offer an improvement of up to 9% in overall hits at the L2 cache level and an IPC (Instructions Per Cycle) speedup of up to 1.08 times that of LRU for a wide range of multithreaded benchmarks.